ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB293

THE TRIP EVENT LOGGER FOR ONLINE FAULT DIAGNOSIS AT THE EUROPEAN XFEL

J. H. K. Timm*, J. Branlard, A. Eichler, H. Schlarb
Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

The low-level RF (LLRF) system at the European XFEL, DESY, is of major importance for high-performant and reliable operation. Faults here can jeopardize the overall operation. Therefore, the trip event logger is currently developped, - a fault diagnosis tool to detect errors online, inform the operators and trigger automatic supervisory actions. Further goals are to provide information for a fault tree and event tree analysis as well as a database of labeled faulty data sets for offline analysis. The tool is based on the C++ framework ChimeraTK Application Core. With this close interconnection to the control system it is possible not only to monitor but also to intervene as it is of great importance for supervisory tasks. The core of the tool consists of fault analysis modules ranging from simple ones (e.g., limit checking) to advanced ones (model-based, machine learning, etc.). Within this paper the architecture and the implementation of the trip event logger are presented.

INTRODUCTION

Fault detection plays an important role in high-performance and safety critical process [1] the operation of particle accelerators. Early detection of faults allows for early intervention, either from the operator himself or automatic intervention from the system. It further avoids fault progression and thus can prevent long down-times. It is not only of interest to detect faults, but also to understand the root of the fault to take the respective measure. And finally, the goal is to proactively prevent faults, e.g., by predictive maintenance. This requires not only the detection and root cause analysis but also a model for prediction.

The prerequisite for early fault detection is to have the appropriate sensor information at run time. In many cases however, it is impossible to relate abnormal behavior seen in one individual sensor signal to the root cause of a fault, as it might be reasoned by different root causes. Furthermore, a fault can quickly propagate through the system leading to a cascade of subsequent faults. Thus, for root cause analysis at best all sensor data involved needs to be monitored jointly and synchronized, and further analysis of the data is necessary to identify the root cause. This analysis can range from a simple limit checking of one single sensor signal to a neural network analysis of all the available data.

With the trip event logger we propose a modular architecture for fault detection and root cause analysis, that allows to easily switch between different of these analysis modules and even having them run redundantly to compare different analysis methods. Further requirements of the trip event logger are that it has to be well integrated in the existing infrastructure. For the European XFEL this requires an integration into the control system DOOCS [2] using the infrastructure of ChimeraTK Application Core [3,4]. This does not only allow to possibly apply to different control systems (e.g., EPICS) but also allows to intervene into the control system, which will be necessary looking towards automatic failure handling and prevention in the future. Further details on the implementation are provided in the next sections.

As discussed above, the trip event logger provides the infrastructure to facilitate deployment of fault detection and analysis tools. In order to also support the development of such analysis tools, the trip event logger needs to be able to be tested offline on recorded data. Therefore, we provide an offline module (using the same code basis) that also includes an adapter to a High Performance Computing (HPC) cluster for efficient analysis. In order to collect a data base with interesting events for further understanding and algorithm development, the possibility to automatically take labeled snapshots is a further desired functionality.

The proposed protocol for fault handling is shown in Fig. 1, for which the trip event logger provides the infrastructure. First, we will discuss this protocol in detail in the next section and report on the implementation details, we want to summarize the intended goal of the trip event logger. The trip event logger should not only support the operator in fault handling or even prevent faults to occur, it should also help to develop the necessary analysis algorithms therefore.

INTERNAL ARCHITECTURE AND PROTOCOL

At the European XFEL, the control system DOOCS [2] is used. Each control system parameter or signal is made accessible in DOOCS and can be assessed by the trip event logger via the fault state base module as shown in Fig. 1 on the far left. These signals can be of different type, e.g., Boolean, integers or time series. The fault state base module contains the different analysis modules, which can all consider different algorithms, monitor different subsystems and focus on different type of root causes. For each analysis module one fault state needs to be defined. Different intermediate states (warnings, etc.) can be defined if needed. If the module is neither in the faulty nor in an intermediate state, it is in normal state. In case of a transition of a state into the faulty state, the state machine triggers a fault report. Then, the trip event report group summarizes all necessary fault information, i.e., the timestamp, the location and the assumed root cause.

* jan.horst.karl.timm@desy.de

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB293

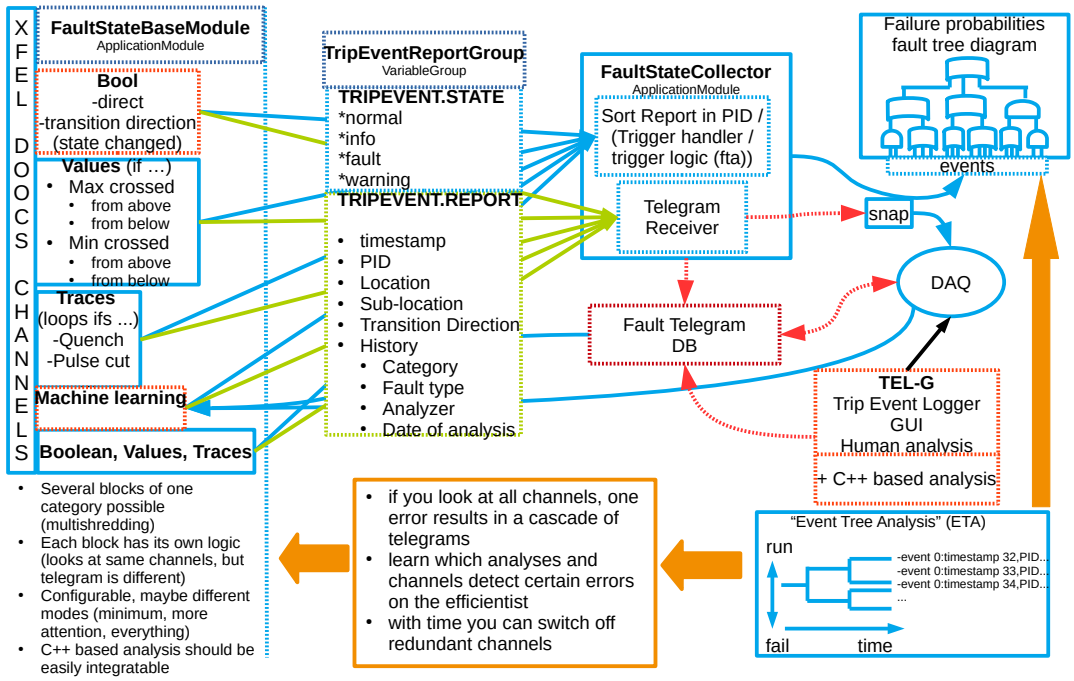

Figure 1: The basic objects of the trip event logger are shown, as well as the protocol for fault detection and for the development and improvement of the root cause analysis.

These reports are then collected in the fault state collector. The fault state collector should also include a trigger for the acquisition of so called snap-shot data for archiving and offline analysis. In an idealized world with exact timestamps, sorting the reports by time should lead to an event tree with the report in the first place defining the root cause. In practice, however, this will not work because, for example, the analyses are not yet sophisticated enough or the timestamps of the various subsystems are not synchronized precisely enough. It is also conceivable that a root cause exists that is not yet known by any analysis module. To address this, an extended analysis is required in the fault state collector that is, for example, trained on insufficient synchronizations between the various subsystems or can also compensate for other deficiencies that have not yet been taken into account. It would be desired if the fault state collector also provides some level of certainty with respect to its event tree analysis, so that for events with low certainty, an expert could go manually through the reports and through the snapshot data. If the real root cause is found, a respective analysis module needs to be developed or adapted accordingly. Thereby the performance of the trip event logger is continuously improved.

IMPLEMENTATION DETAILS

The trip event is implemented in C++ building upon the ChimeraTK Application Module [3]. The objects described in the previous section, e.g., the fault state base module or the fault state collector, are objects in sense of C++ with the corresponding classes inheriting from the ChimeraTK Application Module.

Here only the most important functions of ChimeraTK are mentioned, which are important for this project. The extensive back end of ChimeraTK Application Module provides access to the most common control systems, such as DOOCS, OPC-UA, EPICS, which facilitates the transfer to other accelerators. In addition, the close interconnection to the control system enables feedback to the machine. This is a key requirement with regard to automatic fault handling and prevention, which are the defined long-term goals of the trip event logger. Furthermore, with ChimeraTK Application Module, if the resources of the front-ends allow it, you can run a fault state base module directly on the front-end without the time-delays induced by the data transfer as well as the limitations by the data rate. Moreover, multithreading and scalability are also directly given as these are also basic parts of ChimeraTK.

Figure 2 shows an analysis module in the middle of the figure, which is a standalone object, independent of ChimeraTK. It is used by the trip event logger within the fault state base module for online analysis as well as for offline analysis by the respective tool, called Ladybug. So within the analysis modules some standard functions have been defined which are used by both applications. Ladybug is a collection of tools and libraries to run the event analysis also offline over the snap-shot data with the possibility to use an HPC cluster. The tool is provided within a Docker container that contains all the required dependencies, including those of the analysis module. The container can be built from a Dockerfile, so dependencies can be easily added, or removed. The core program, Ladybug itself, automatically recognizes various data formats, such as HDF5 and

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-WEPAB293

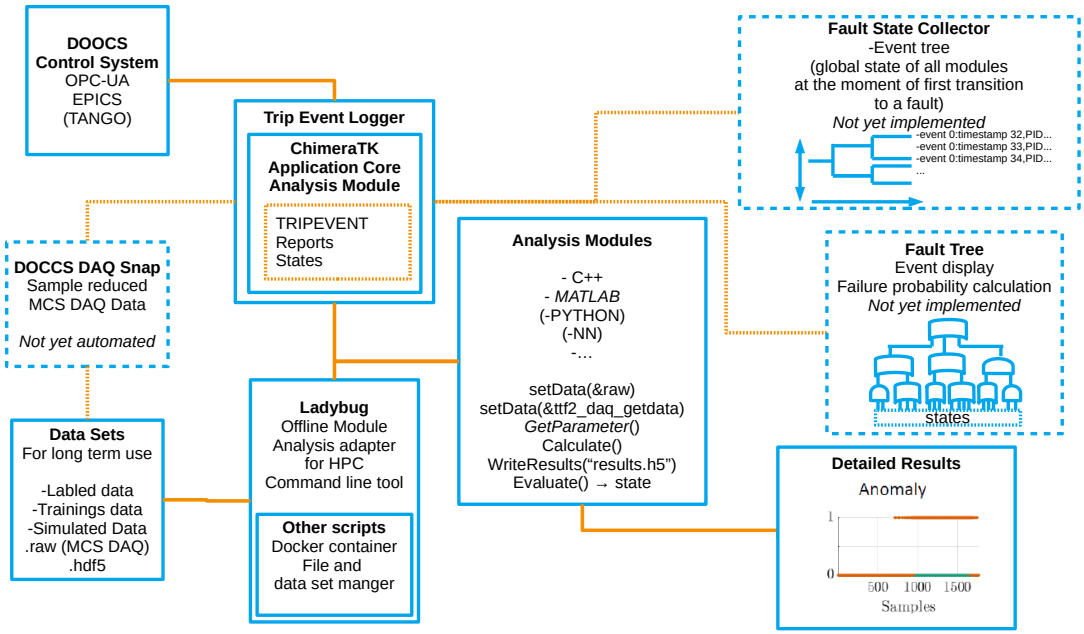

Figure 2: General overview of the trip event logger and the infrastructure.

the DOOCS internal raw format. It can also start the event analyses with different options, so that, e.g., a system identification can be made. In the case considered here, which is explained in more detail in the next section, this system identification requires a grey-box identification, but the same functionality could also be used for example for a neural network training, if the analysis module is based on some neural network model. There is also a simple data manager available, which can package the data sets into small packets. These can then be used by bash and slurm scripts to submit jobs to the HPC cluster jobs.

FIRST RESULTS AND CONCLUSION

The first implemented analysis module is a model-based method based on the parity space [5]. It focuses on the anomaly detection of superconducting radio-frequency (SRF) cavities. This is of practical importance as the SRF cavities as heart of the LLRF system at the European XFEL are of major importance for a high-performant and reliable operation. Essentially, the parity space method looks at six signals for each cavity, being the forward, reflected and probe signals in amplitude and phase. Based on these signals it can detect abnormal behavior. We demonstrated that a cavity can be observed online in this way. Furthermore we verified that the monitoring can easily be extended to further cavities. Here, the data transfer rate is the bottleneck that limits the number of cavities per workstation due to high sampling rate of 9.027 MHz within a pulse, where the pulse rate is 10 Hz.

With regard to the offline analysis of snap-shot data, the provided tools leaded to a significant improvement with respect to calculation time. For example, with the described infrastructure in place, data of two weeks from 32 cavities was processed within hours on the MAXWELL HPC cluster with standard resources, which previously took about a month on a local workstation.

In Fig. 2, the parts shown within dashed boxes have not been implemented yet. This will be subject to future work on the architecture side. Further analysis modules need to be developed. This requires substantial expert knowledge from the areas of accelerators, fault detection but also machine learning and HPC expertise. However, with this work an infrastructural basis for such developments could already be created, which facilitates the development and deployment.

ACKNOWLEDGEMENTS

The authors acknowledge support from DESY (Hamburg, Germany), a member of the Helmholtz Association HGF. Special thanks go to Nicholas Walker, DESY, and Mathieu Omet, KEK, for their inputs.

REFERENCES

[1] R. Isermann, *Fault-Diagnosis Systems*, Berlin, Germany: Springer, 2006.

[2] Distributed Object Oriented Control System (DOOCS), https://doocs-web.desy.de/.

[3] G. Varghese *et al.*, "ChimeraTK - A Software Tool Kit for Control Applications", in *Proc. 8th Int. Particle Accelerator Conf. (IPAC'17)*, Copenhagen, Denmark, May 2017, pp. 1798–1801. doi:10.18429/JACoW-IPAC2017-TUPIK049

[4] ChimeraTK Repository, https://github.com/ChimeraTK

[5] A. S. Nawaz, S. Pfeiffer, G. Lichtenberg, and P. Rostalski, "Anomaly Detection for Cavity Signals - Results from the European XFEL", in *Proc. 9th Int. Particle Accelerator Conf. (IPAC'18)*, Vancouver, Canada, Apr.-May 2018, pp. 2502–2504. doi:10.18429/JACoW-IPAC2018-WEPMF058